Application Development Domain

Technical Architecture

February 6, 2003

Version 2.4

History of Changes

12/06/2000	Added a Too Be Determined List (GAPs)
2/21/2002	Standards 1: wireless & mobile devices added. Standards 2: VB.NET added as Research/Emerging. Standards 4: Use of XML for parameter and configuration files added Standards 5: Use of XML documents for message passing added. Standards 5: Web Services, W3C XML Schema, and XLST added as Research/Emerging. Standards 6: ASP.NET added as Research/Emerging. Standards 6: Table added comparing Java Server Pages vs. Active Server Pages. Standards 6: Java / Visual Basic Integration – Web Service added for consideration. Standards 6: Depoyment Guidelines – Information on deploying Internet applications in DMZs added; deployment of Intranet/Extranet Applications using Java Web Start added. Standards 6: Integrated Development Environments (IDEs) – JBuilder, Visual Studio.NET, XMP SPY 4.0 added as Research/Emerging; footnote clairfying the role of Visual C++ with Visual Basic added. Standards 7: Application Security Standards – replaced by appendix A, Application Security Guidelines. Standards 8: Standard Project Management Tools – PlanView PSA added as Research/Emerging. Standards 11: Developer-Oriented Report Writers – Actuate e.Reporting Suite 5 added to Research/Emerging. To-Be-Determined List – Modified to add Web-based Ad Hoc Tools, and Web-based OLAP tools; consideration of OO COBOL as a strategic language was dropped. Appendix A, Application Security Guidelines added.
5/21/2002	Standards 2: Standard Business Tier Lanquages - Oracle Forms (PL/SQL) listed as Transitional. Standards 6: Java Server Pages (JSP) with Servlets and Enterprise Java Beans – Oracle Web removed from Research/Emerging. Standards 6: Intranet/Extranet Applications – Deployment of internal Oracle Forms applications via a browser using Oracle 9iAS listed as an acceptable interim web-enabling technique until applications can be migrated to Java. Standards 6: Integrated Development Environments – Oracle Forms

	moved from Research/Emerging to Transitional.
6/25/2002	Standards 11: Developer-Oriented Report Writers replaced by three standards sections. Standards 11: Enterprise Reporting / Structured Information Delivery; Standards 12: Ad Hoc Query / Analysis; and Standards 13: Online Analytical Processing (OLAP) tools.
10/18/2002	Standards 2: Added caveat to avoid using proprietary features or extensions to Java. Standards 5: Use of XML Schema and XSLT reclassified from research to strategic.
2/6/2003 <NEW>	Standards 2: Added mandatory compliance to State Java Coding Standards and Conventions for custom-developed Java applications. Standards 14: Geographic Information Systems (GIS) Software Standards added.

Table of Contents

Mission

Application Architecture identifies criteria and techniques associated with the design of applications for the State's distributed computing environment that can be easily modified to respond quickly to the State's changing business needs, as well as to the rapidly evolving information technologies available to support those needs.

Introduction and Background

The State of Connecticut, like most large public and private enterprises, relies heavily on computer applications to support its business operations. Because the State's business processes change dynamically in response to both legislation and new demands from citizens, it is important that the State's computer applications also be able to change rapidly.

However, many current State applications are either monolithic or two-tier client/server applications. This existing inventory of "legacy" applications reflects the tools available at the time the applications were developed and how system development projects were funded and managed. As illustrated in Figure 2-1, many applications were designed and funded to perform a specific operation for a specific agency on a specific hardware platform.

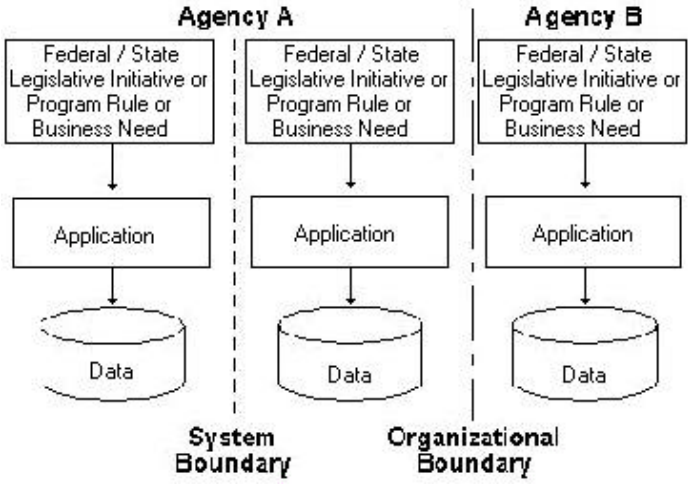

Figure 2-1. Legacy Applications were developed and operated independently.

These applications typically were developed independently using different languages and tools. The ability to communicate with other applications or systems or to adapt to changes in the business processes generally was not a design requirement.

The architecture of the existing inventory of applications adversely impacts the State's business in four key ways:

- The cost and time needed to modify existing applications to support new business requirements
- The cost and time required to enable existing applications to take full advantage of rapidly evolving networking technologies such as the Internet or new networked systems such as GIS (Geographic Information Systems)

- The difficulty in integrating applications to share common services and data
- The expense to develop, use, and maintain new applications because there is little reuse of code between applications

Recently, application development tools and technology have begun to evolve to help address these problems. A number of options now exist to meet business needs and deliver information to people when and where they need it. These options include:

- Reuse of Code: Units of code previously duplicated in many applications can be packaged into components or services for reuse in different applications.
- Middleware: Shared software allows applications to communicate with each other, access data residing on different platforms, and access shared services.
- New User Interface Options: There is an expanding array of user interface options - including Web browsers, personal digital assistants (PDAs), and interactive voice response units (IVRs).

Using these components in well-designed three-tier or N-tier, client/server application architectures can create solutions to meet the State's ever changing business needs.

Here are some examples that address issues facing the State today.

- If the code used in many applications for basic e-mail functionality was implemented as a service in a shared component, we could change e-mail technology without affecting the applications using this service.
- Using middleware, an application to issue fishing licenses could access an application verifying child support payments, even if the applications are developed and run on different types of systems for different agencies.
- When an inmate convicted of certain offenses is released from prison, a Web-enabled application and extranet supporting the prison release process could automatically issue notices to local authorities. The application would be proactive, "pushing" information to users rather than the users "pulling" information out of databases to use it.
- State Highway Patrol personnel equipped with GIS capable PDAs could determine the current location and best routing for emergency response vehicles.

There are many ways applications can be designed to maximize their flexibility, including:

Logical application boundaries. Applications should be designed along logical application boundaries that mimic the business processes they support (Figure 2-2 and the following text).

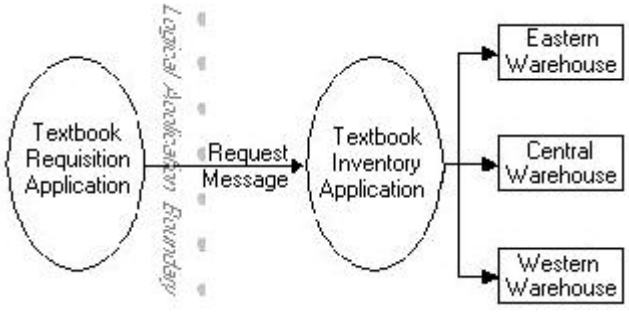

Figure 2-2. Logical Application Boundary

Suppose the State establishes regional textbook warehouses to supply books to the public schools. A school places an order using the textbook requisition application that sends an order message across the logical application boundary to the textbook inventory application. The inventory application decides which warehouse should ship the books based on its proximity to the requester and the availability of the requested textbooks.

The requisition system does not need to know that there are multiple warehouses. This would allow changing warehouse locations (e.g., add a new warehouse or consolidate warehouses) without disrupting the requisition process. The requisition application delegates responsibility to the inventory application to route the order to the most appropriate warehouse.

Asynchronous processing. Applications can be designed to take advantage of new methods of communication involving asynchronous processing.

Just as voice mail permits communication without both parties being available at the same time, asynchronous messaging technologies permit the "de-coupling" of related applications. An application does not have to batch requests before sending them to a second application, nor do both applications have to "standby" while preparing, sending, processing and responding to requests. The applications are designed to put requests and responses in a queue. (See Figure 2-3.) Each application has the flexibility to process the information when it is ready (e.g., when information is generated, after an application completes its current job, or at the next logon).

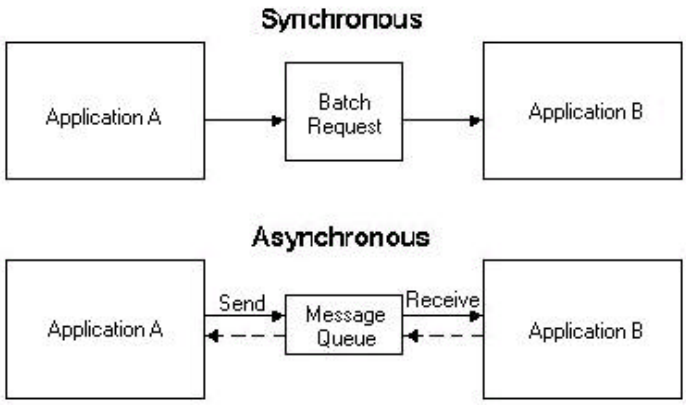

Figure 2-3. Asynchronous Processing

Asynchronous processing is especially critical as:

- The number of remote and/or mobile users with on-demand dial-up or wireless communications increases; and
- Flexible, high performance, shared networks (LANs/WANs/ Internet) become involved that assure very high availability of all resources, but may not provide the constant delay times for the communication needed for traditional synchronous (e.g. SNA) applications.

Using components. Designers can build flexible, scalable, and extensible applications by using components as application building blocks, similar to building cars on an assembly line. Using previously built and tested components in different ways or with new components can accelerate the design, development, and delivery of new applications. Sharing of components across applications also can eliminate significant duplicate design and test efforts. (See Figure 2-4.)

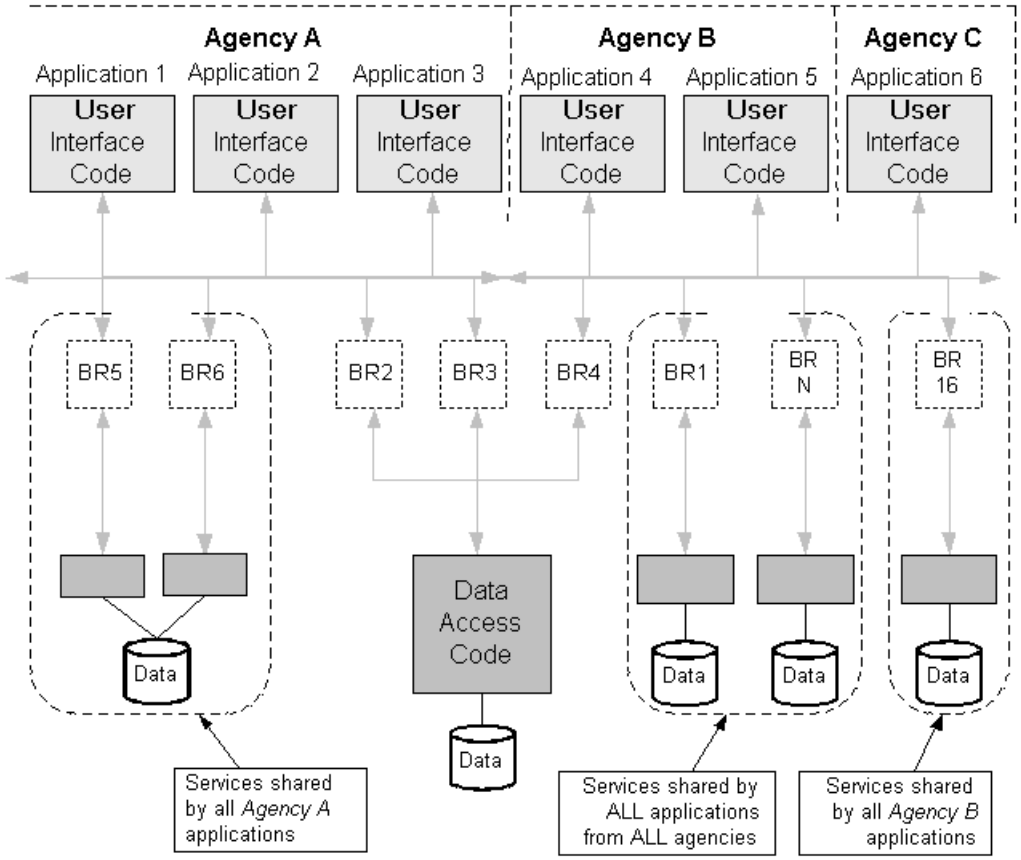

Figure 2-4. Using Components

With this approach, each component addresses one business rule. If the business rule changes, only that component needs modifying. The rest of the application remains unaffected. Equally important, the business rule is changed only once regardless of how many applications use it.

The greatest efficiency can be achieved by combining new technologies (e.g., web-enabled applications, middleware and components), applications designed for flexibility, and methods that foster a culture of reuse. To do this effectively, developers' functions may need to change as new roles evolve within the State's IT organizations for technicians having specialized skills to:

- Identify, analyze, and understand key business processes.
- Design, develop, test, and maintain components.
- Architect applications that effectively deal with multiple user access methods, as well as with various local area network (LAN) and wide area network (WAN) performance constraints.
- Optimize inter- and intra-application communications in terms of both messaging formats and message exchange mechanisms.

The technical topics in this chapter describe designing, developing, and managing applications in more detail, including standards and best practices that apply to those topics. The Technical Topic on designing and developing applications compares the different architectural patterns such as monolithic, n-tier client/server and web-enabled applications. It also details some of the benefits of developing three-tiered and component-based applications.

Application Architectures

Designing And Developing Applications

Introduction

All computer applications have three general areas of functionality:

Interfaces allow applications to communicate with users, other applications, and data resources. In n-tier applications, changes in business rules normally do not require changes in interface code. Interfaces may need updating for other reasons. Examples include when changes occur in another computer system that interfaces with that application or when users need a graphical user interface instead of a character-based interface for that application.

Since applications interface with people, the user interface receives the most attention. Other interfaces are equally important. Traditionally, people accessed computer applications using character terminals (e.g., 3270) or graphical user interfaces (e.g., Microsoft Windows). Recently introduced interfaces include telephones (via IVRs), web browsers, and wireless devices.

Business rules support the business processes that agencies follow. The rules automate the process, define what must be done, and how it must be done. As agency business processes change, the business rules in the applications that support the agencies also must change. Business rules can be isolated into components.

Two examples of State business rules would be:

- Issue a check IF (a) an invoice has been presented AND (b) the invoice is for work for which a purchase order was issued AND (c) the work has been performed AND (d) there is enough money in the bank to cover the check.
- This student is eligible for early graduation IF (a) (s)he has completed the required work AND (b) (s)he has achieved a grade point average of 3.0 AND (c) it is not yet time for her/him to graduate AND (d) (s)he is at least 16 years old.

Business rules are processes followed when business events occur (i.e., business events are triggers for business rules). If business rules define what to do, business events define when it should be done. The following business events might invoke the associated business rules:

- A person applying for public assistance triggers the business rules for “Determine eligibility for public assistance.”

A person applying for a corporate charter triggers the business rules for “Process application for incorporation.”

- A motorist driving erratically triggers the business rules for “Traffic stop.”
- It is April 16 triggers the business rule for “Late tax return.”

Data access. Data access code automates the storing, searching, and retrieving of data by computer applications. In n-tier applications, changes in business rules may not require changes to the code that accesses data, but occasionally, they do.

The ways in which these application functions are assembled determines:

- The flexibility of the application.
- How quickly the application can be modified to support changes in business and technology.

- How easily the application interfaces with people and with other applications.

Middleware provides links among the functional levels of the application, as well as with the components supporting the application.

- Application architecture should be independent of any specific technology or set of development tools. Its components, interfaces, business rules, and data access code, can be implemented with any standard language on any supported platform that supports the business needs of the application.

Application development tools are critical to the development and support of applications. Regardless of the tools used, it is important to design each tier to be portable across platforms. Tool limitations can, however, impact tradeoffs in an application's design/architecture. The architecture should determine the tool selection, not the other way around.

There are three approaches for selecting tools to develop client/server applications:

Best of breed. Use separate, specialized tools for each application tier. Use middleware to support communications between the different tiers.

Front end/back end. Two different tools are used: a specialized user interface development tool and an integrated tool set that provides middleware for the business rules and data access tiers. The middleware must support communications between the user interface and other two tiers.

Integrated. Integrated tool sets, or CASE tools, are used that generate code for all tiers of the application. These tools provide the middleware necessary to support communications between all application tiers.

N-tier service oriented application architectures require additional types of tools.

- Repositories (libraries) to keep track of business rules that have been automated by components.
- Software management tools to provide version control, configuration management, and software distribution services.

Application design and development consists of the following technology components: Monolithic Applications, Two-Tier Client/Server Applications, Three-Tier Client/Server Applications, N-tier Service-Oriented Applications, and Web-Enabled Applications. These components will be discussed in a historical sense.

Application architectures

Application Architecture 1: Monolithic Applications

Monolithic applications are applications where the code that implements the business rules, data access, and user interface are tightly coupled together as part of a single, large computer program. A monolithic application typically is deployed on a single platform, often a mainframe or midrange computer. There are examples of monolithic applications running on smaller systems - or even distributed across multiple machines. The determining characteristic of a monolithic application is that the code is tightly coupled and highly interdependent.

Monolithic computer applications are deployed across the State. Since the State provides many different services to its citizens, there are many computer applications to support those services. In most cases, these applications were developed independent of each other using different

combinations of technology. For example, one agency application may use COBOL, CICS, and VSAM. Another application to support the same group of citizens may use COBOL and IMS.

Monolithic applications have several drawbacks:

- It is costly and time consuming to modify monolithic applications.

Changing one piece of code that implements a business rule, accesses data, or provides an interface to users or other systems likely impacts other code in the application. When any code in a monolithic application changes the entire application must be re-tested and re-deployed.

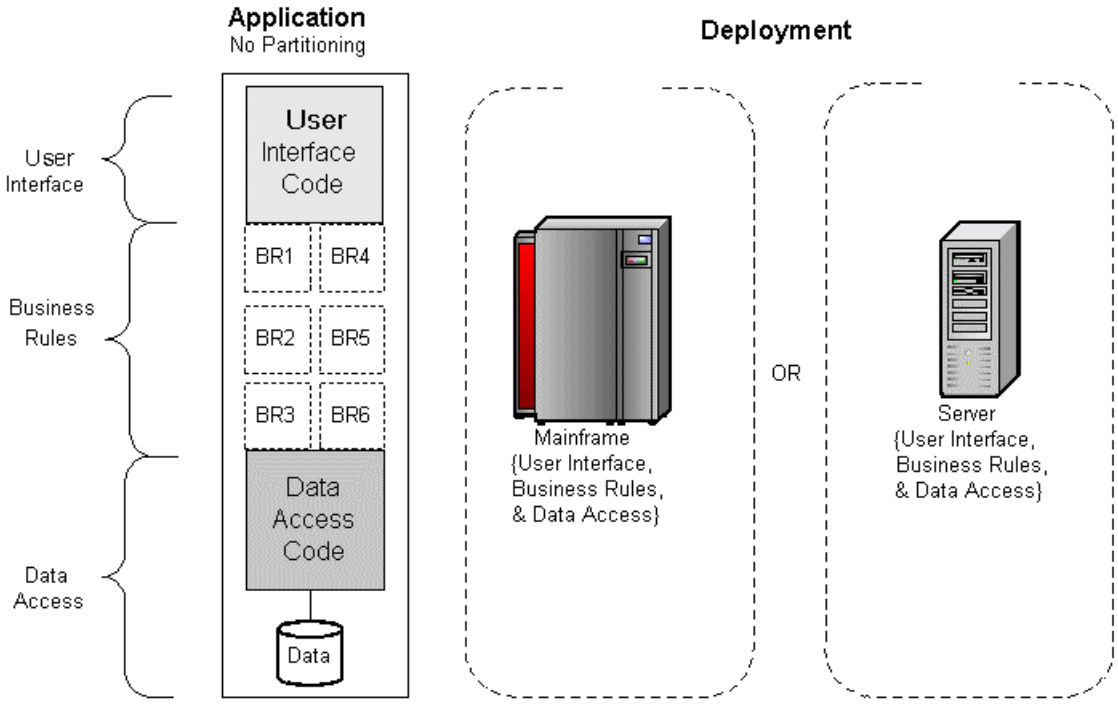

Figure 2-5. A monolithic application

- It is difficult to integrate monolithic applications to share services and data.

  Most monolithic applications do not have well-defined interfaces that can be accessed by other applications or new user interfaces.

- There is little reuse of redundant code between monolithic applications, making it more expensive to build and maintain them.

  Many monolithic applications contain functionality already replicated in other applications. Monolithic applications are slower and more costly to build because existing functionality must be reinvented many times. Monolithic applications are more expensive to operate, since the same data often has to be gathered, entered, and stored in many places.

- It is difficult to have monolithic applications communicate with other applications.

  Most existing applications do not have the ability to communicate with other applications, within an agency, and with applications in other agencies.

- Monolithic applications can be accessed using only a single user interface.

Most monolithic applications were developed to be accessed via 3270 terminals. Having a single user interface is a limitation when application services need to be accessed from other user interfaces such as Web browsers or the telephone (via IVRs).

- There is little flexibility where monolithic applications can be deployed.
  Most monolithic applications must be deployed on a single machine type, for example a mainframe. This could be because either the software code is tightly coupled to that machine type or the mainframe is needed to get enough processing capacity to process all parts of the application: the user interface, the business rules, and the data access code.

Application architecture 2: Two-tier client/server applications

Some State agencies have attempted to overcome the business impact of monolithic applications by adopting client/server technology for new applications. The terms “client/server”, “client”, and “server” are often misunderstood. Many believe that “client/server” means an application with a graphical user interface and a relational database. Neither is necessarily true. In fact, client/server applications are constructed of software “clients” that, in order to perform their required function, must request assistance - “service” - from other software components known as “servers.” Middleware provides communication between the client and server.

Early client/server applications used architectures dictated by the tools used to write them. As a result, most early applications used a *two-tier client/server architecture*. The “tiers” of client/server applications refer to the number of executable components into which the application is partitioned, not to the number of platforms where the executables are deployed. Sometimes, the tiers into which the application is partitioned is called “logical partitioning”, and the number of physical platforms on which it is deployed is called “physical partitioning.”

In two-tier client/server architecture, application functionality is partitioned into two executable parts, or “tiers.” On one model, one tier contains the code that implements a graphical user interface (GUI) and the code that implements the business rules. This tier executes on PCs or workstations and requests data from the second application tier, which usually executes on the machine where the application’s data is stored. This model is referred to as *two-tier, fat client*. Though while the application has two tiers of executable code, most of the code is contained in the tier executing on the workstations - the “fat client.” (See Figure 2-6)

Since business rules are tightly integrated with user interface code, the code that implements the business rules must be deployed on the same platform(s) as the user interface. Thus, the entire workstation-resident portion of an application must be re-deployed if a business rule or the user interface changes. If the number of workstations is high or the workstations are geographically dispersed, the maintenance costs for two-tier, fat client applications can escalate quickly.

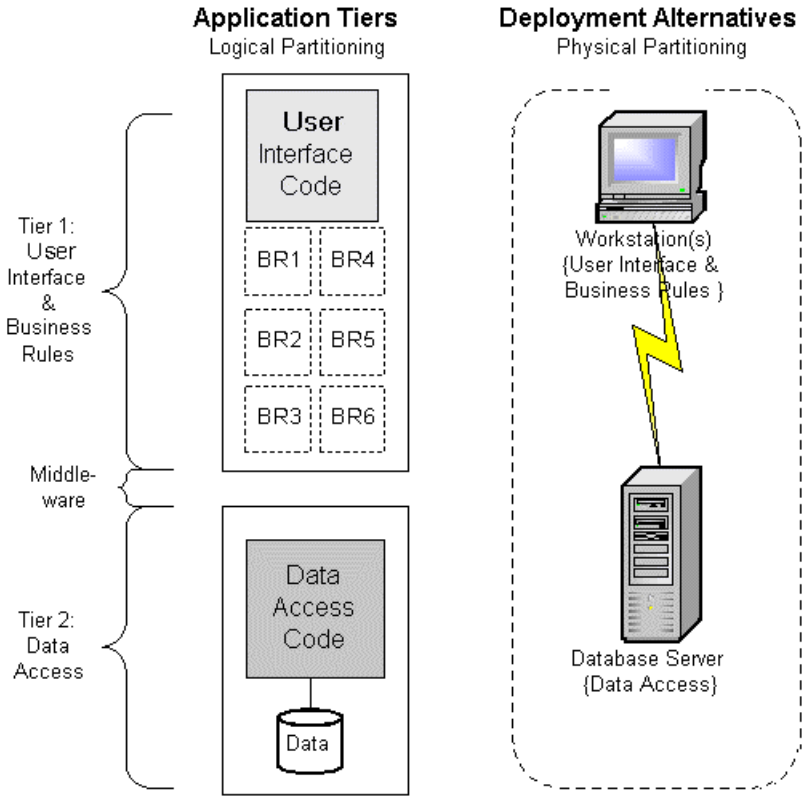

Figure 2-6. A two-tier, fat client application

A second model for two-tier client/server applications has much of the code that implements the business rules tightly integrated with the data access code, sometimes in the form of database stored procedures and triggers. This model is called *two-tier, fat server*. (See Figure 2-7.)

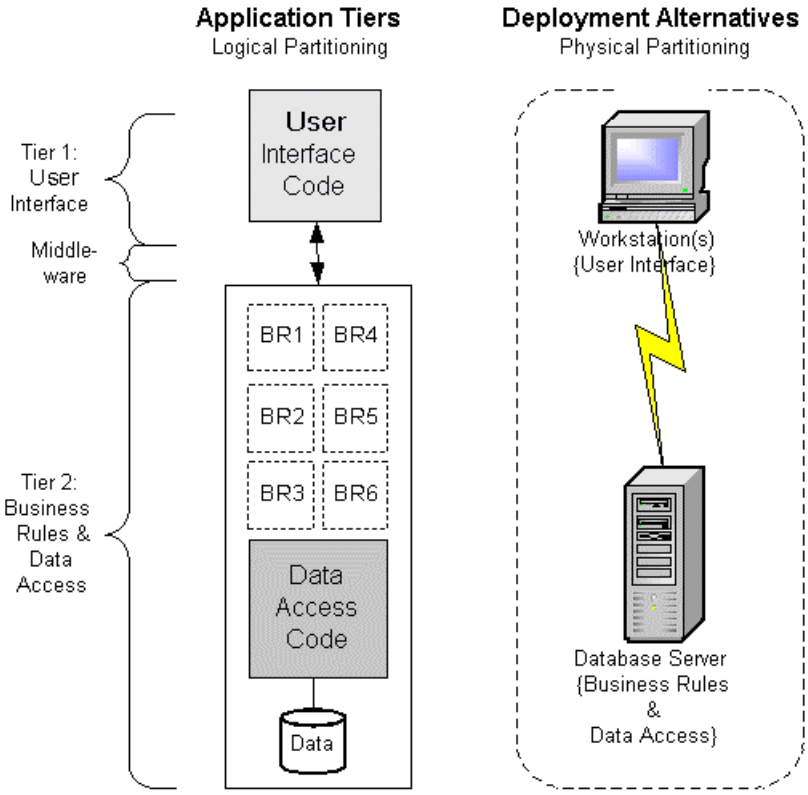

Figure 2-7. A two-tier, fat server application

Two-tier, fat server applications are often implemented as mainframe applications with Web browsers as user interfaces. This approach may be a useful first step to migrate to a three-tier or n-tier service oriented application architecture. Users can enjoy the speed and ease-of-use provided by the web's graphical interface while developers update other parts of the application.

Since the business rules in two-tier applications are tightly integrated with the user interface code or data access code, two-tier client/server applications have the following drawbacks:

- Two-tier client/server applications are difficult and expensive to modify when business requirements change.

  The business rules tend to be monolithic. Changing a business rule may impact other business rules and the rest of the application.

- There is little reuse of redundant code in two-tier client/server applications.

  It is difficult to reuse business rules elsewhere (e.g., in other computer applications that require similar services or in batch processing that is part of the same application) when they are tightly coupled to each other and to the user interface (fat-client) or the data (fat-server).

- There is little flexibility in selecting the platforms where the two-tier client/server applications will be deployed.

  In two-tier, fat client applications, the business rules must execute on the same platform as the user interface because the code they are implemented in is tightly coupled with the interface. Likewise, in two-tier, fat server applications, the business rules can only execute on

the machine that hosts the database because they are implemented either with or inside the database.

- Users only can access two-tier client/server applications with PCs running a graphical user interface.

  Since the user interface is graphical and requires a workstation, users with other I/O devices are excluded from using the application. These devices include existing non-graphics terminals (e.g., UNIX terminals or 3270 terminals), telephone interfaces via IVRs, and new user interface devices still evolving (e.g., PDAs and other mobile communications devices).

- Two-tier client/server applications can be more difficult to manage than monolithic applications.

  Changes to either business rules or the GUI often mean that the entire workstation-resident portion of the application must be redistributed and reinstalled on every workstation that uses the application. Such software distributions can be time-consuming, costly and logistically difficult to manage.

Application architecture 3: Three-tier client/server applications

Three-tier client/server applications are partitioned into three executable tiers of code: the user interface, the business rules, and the data access software. This does not mean that the three tiers execute on three different platforms. Often, the business rule tier is deployed on the same platform as the data access tier, or on the same platform(s) as the user interface.

Properly implemented three-tier client/server applications can achieve higher performance efficiency by providing more flexibility in where application executables can be deployed as well as by making use of an enterprise's n-tier shared services. This makes three-tier client-server applications a good transition step from monolithic or two-tier applications.

Figure 2-8 illustrates a three-tier client/server application. Notice that in the deployment - or physical partitioning - of the application, the business rules are separate from the user interface and the data access code. The business rules may be deployed on their own server or on the same server as the database. Although it is also possible to deploy the business rules on the same platform as the user interface in a three-tier architecture, it is not recommended because of the software management problems associated with using many or dispersed user workstations.

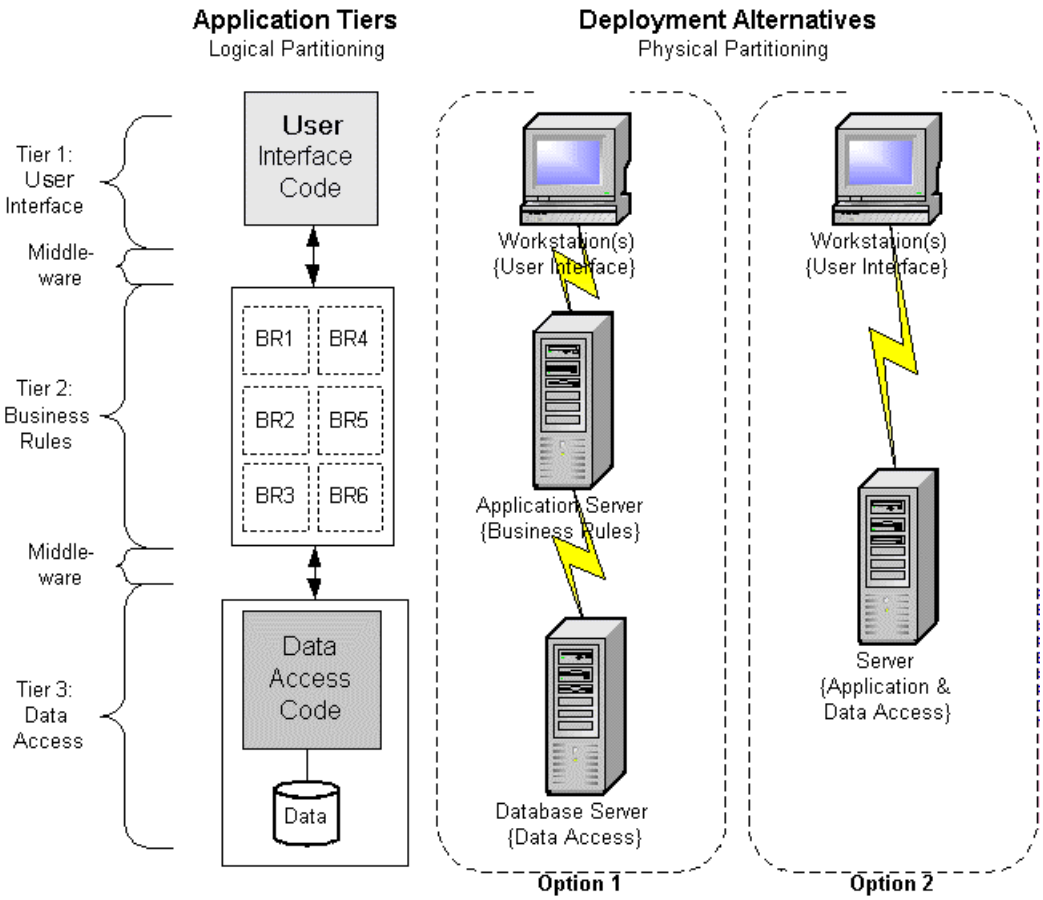

FIGURE 2-8. A THREE-TIER CLIENT/SERVER APPLICATION

Three-tier client/server applications offer the following advantages:

- Three-tier client/server applications can be easier to modify to support changes in business rules.
- With three-tier client/server applications, there is less risk in modifying the code that implements any given business rule.
- Three-tier client/server applications can be made to support multiple user interfaces: character, graphical, web browser, telephones, and others.

Application architecture 4: N-tier Service Oriented Architecture

Many problems inherent in the State's existing monolithic and two-tier applications can be overcome by implementing applications with a three-tier architecture. However, large, complex projects that are anticipated to have high usage volumes and/or long life spans may be better served by an n-tier service oriented architecture.

In the n-tier service oriented architecture, applications are partitioned into discrete functional units called "services." Each service implements a small set of related business rules or function points. If a business rule must be modified to support changing business requirements, only the

service that implements that business rule is impacted. The remainder of the application remains intact.

The adaptability of applications is further enhanced by the use of an n-tier shared services architecture that segments rule processing into a series of services that can be accessed individually. In the application illustrated in Figure 2-9, each business rule is implemented as a discrete executable (a "service") that any client can request.

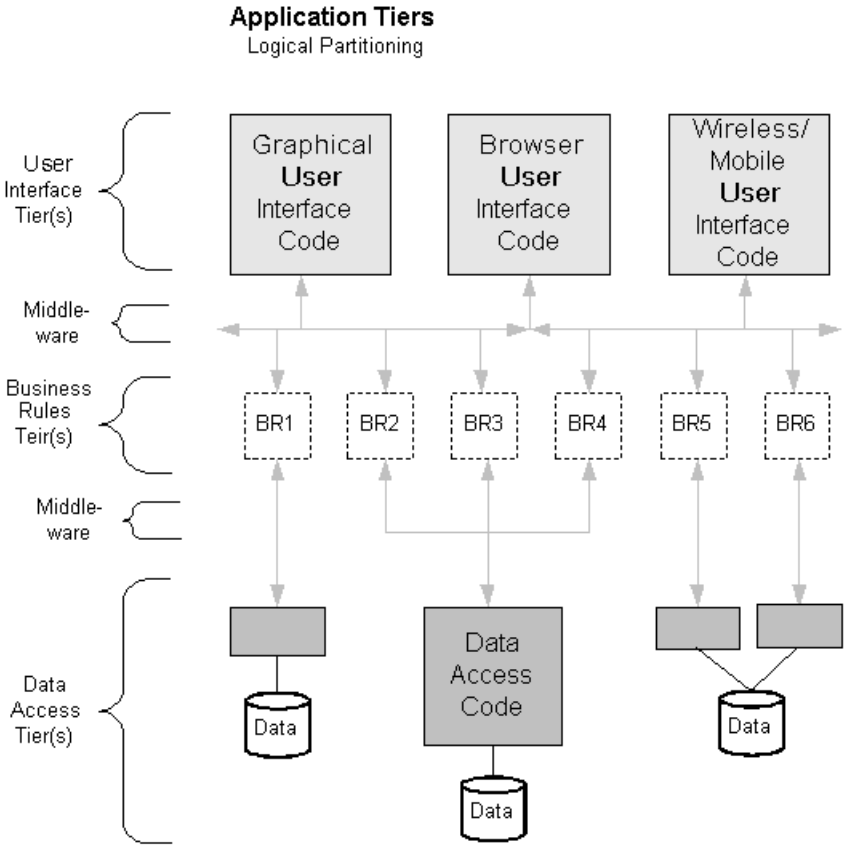

Figure 2-9. An n-tier client/server application

The maximum benefits of n-tier architecture are realized when many n-tier applications are deployed across the State, sharing common software services and offering multiple user interfaces. In this environment, any application can access any service, provided the application has the proper security permissions. In n-tier service-oriented application architecture:

- Some services will be shared by applications from multiple agencies.
- Others services will be shared by applications within a single agency.
- A few, highly specialized services may be developed, at least initially, for a specific application.

Since the business rules are implemented as separate executables, any combination of business rules may run on any combination of platforms. This offers flexibility in selecting the platforms where the application components can be deployed, resulting in a high degree of scalability. As transaction loads, response times, or throughputs change, an individual service can be moved from the platform on which it executes to another, more powerful platform.

Since business rules are implemented discretely instead of being tightly integrated with the graphical user interface, changes to business rules typically do not require updates of code on the workstations accessing the application. This is very important in managing an application with many, geographically dispersed workstations.

Also, since business rules are implemented in discrete services, the same business rule can be invoked by users accessing the application from a GUI, from character terminals, from web browsers, by telephone from IVRs or by batch jobs. A separate interface tier provides programmer productivity and consistency of application behavior.

N-tier service oriented applications offer the following key advantages:

- N-tier service oriented applications are highly scaleable.
- An n-tier service oriented architecture offers the best performance of any client/server.
- N-tier service oriented applications offer the highest potential for code reuse and sharing.

The greatest strength of a service-oriented architecture is the opportunity it provides for the repeatable, rapid development of new applications. Figure 2-10 illustrates N-tier applications in a service-oriented architecture.

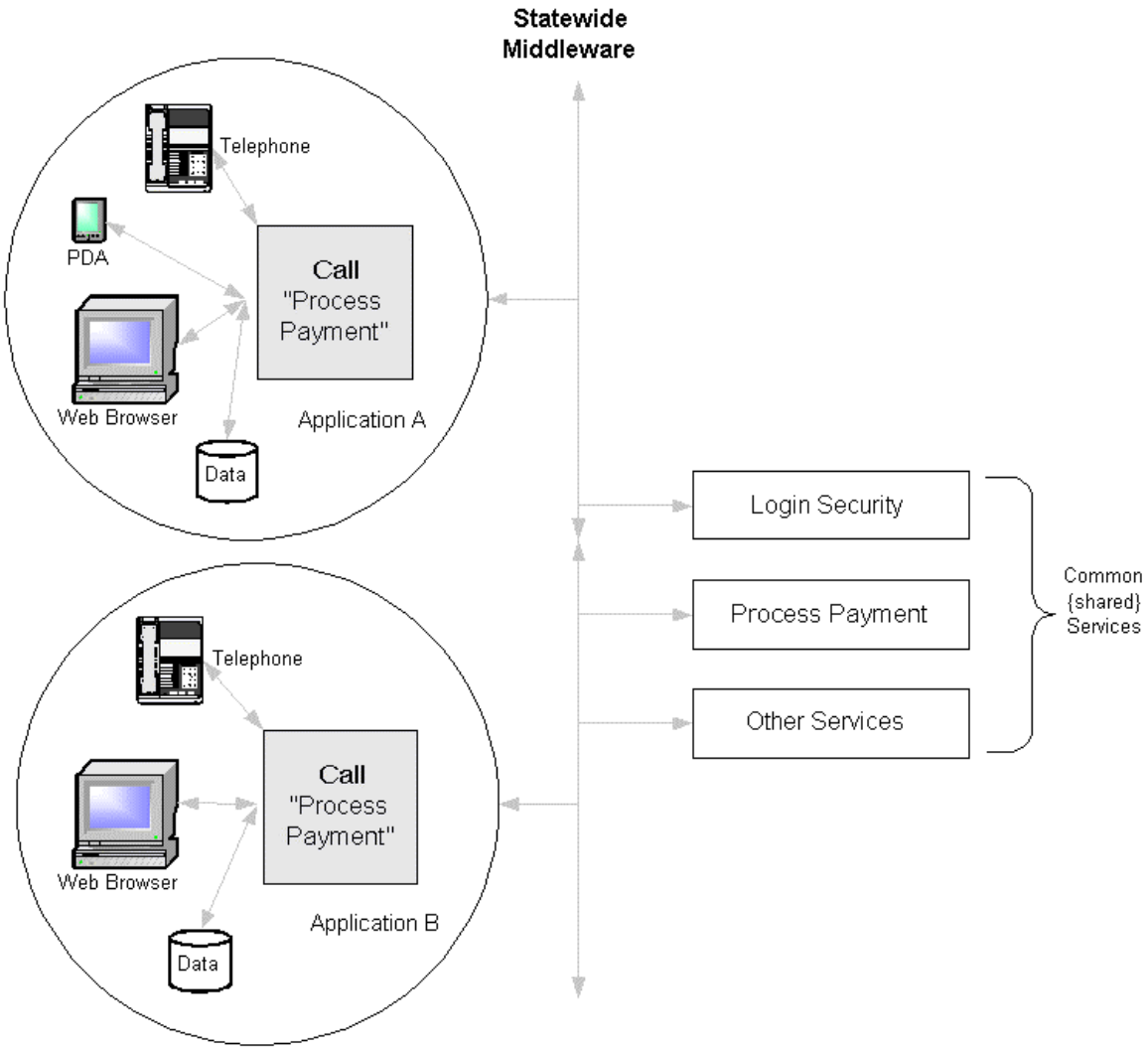

Figure 2-10. N-tier applications in a service-oriented architecture

Application architecture 5: Web-enabled Applications

There are two types of web-enabled applications. Some web-enabled applications provide information to clients in page format using HTML and XML to manage content dynamically. Other Web-enabled applications have fully interactive functionality and near real-time transaction processing capabilities.

Web-enabled applications are a special case of client-server applications where the “client” is a standard Web browser like Netscape Communicator or Microsoft Internet Explorer. The browser serves as another type of user interface (thin client) in the three-tier or n-tier application. Use of a standard Web browser as the client provides the user with a familiar, intuitive interface and significantly simplifies the process for developing and distributing the user interface.

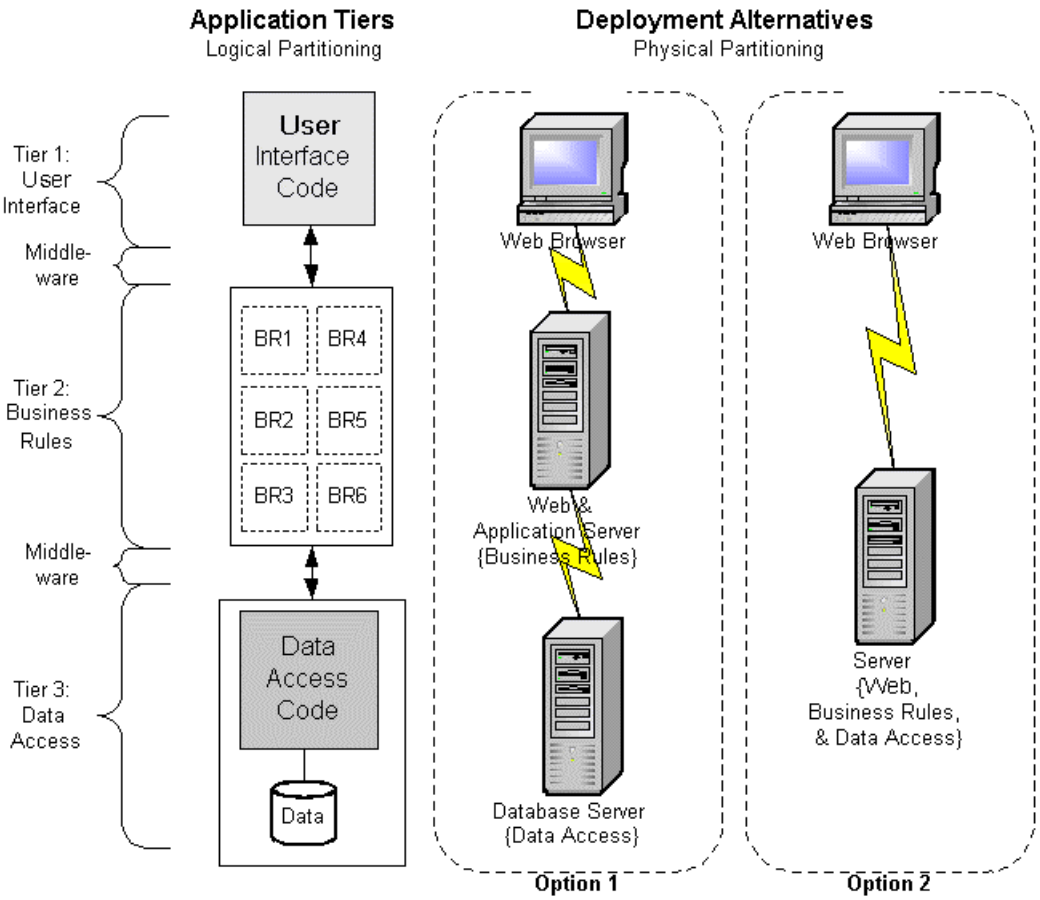

Figure 2-11. Web-enabled applications use a Web Browser for the user interface

Ideal web-enabled applications for the State are n-tier service oriented applications that use:

- An industry standard Web browser as the thin client;
- Intranets to provide secure access by State users;
- Extranets to provide restricted access by selected State business partners; and
- The Internet and firewall technology to provide managed access by citizens and other interested parties.

Web-enabled applications will continue to grow in importance as a means to timely and cost effective delivery of information to the State's employees, business partners and citizens.

Web browsers are applications that accept text in the form of HTML/XML statements. The HTML/XML is interpreted and the file is presented on the desktop screen in web page format based on the corresponding HTML/XML. Web pages can contain hyperlinks to other documents, and multimedia such as text, images, audio and video.

The web started out as an environment for publishing static pages using HTML. Early on, the notion of enabling interactive, transaction oriented applications via the same browser became attractive since it could eliminate the need to install client software on every user's workstation. Browser technology supports the execution of programs written in scripting languages embedded in an HTML page. Browser technology also supports the execution of programs written in

scripting languages including JavaScript, VBScript and others. Browsers may also support the running of Java Applets in the context of a Java Virtual Machine (may require a plug-in).

Principles

Principles are intended to guide the evaluation, selection, design, construction and implementation of this domain and its elements.

Principle 1. Information is an enterprise asset

Information is valued as an enterprise asset, which must be shared to enhance and accelerate decision making.

Justification

- Enhances the efficiency and effectiveness of the delivery of services.
- Most information is in isolated pockets such that the value of information is not always recognized.
- Enables new enterprise-wide solutions.
- Treating data as an enterprise asset increases its integrity and relevance.

Implications

- Need consistent data dictionaries, general-purpose objects, and component reuse across agency applications.

Principle 2. Leverage Data Warehouses

We should leverage data warehouses to facilitate the sharing of existing information to accelerate and improve decision-making at all levels.

Justification

- Data can be replicated and combined from multiple agencies without changing the originating systems or developing new systems.
- Reduced business cycle times have led to a need for faster access to more information.
- There is a significant burden on programmers to generate reports and data queries. Data warehouses and their associated end-user tools make it possible to relieve this burden by making it the responsibility of end users.
- Warehouses fulfill the need for internally consistent data.

Principle 3. Ensure Security, Confidentiality and Privacy

IT systems should be implemented in adherence with all security, confidentiality and privacy policies and applicable statutes.

Justification

- Helps to safeguard confidential and proprietary information.
- Enhances public trust.
- Enhances the proper stewardship over public information.
- Helps to ensure the integrity of the information.

Principle 4. Integration

Systems must be designed, acquired, developed, or enhanced such that data and processes can be shared and integrated across the enterprise and with our partners.

Justification

- Increase efficiency while better serving our customers (e.g., the public, agencies, etc.).
- Redundant systems cause higher support costs.
- Ensures more accurate information, with a more familiar look and feel.
- Integration leads to better decision making and accountability.

Principle 5. Reduce Integration Complexity

The enterprise architecture must reduce integration complexity to the greatest extent possible.

Justification

- Increases the ability of the enterprise to adapt and change.
- Reduces product and support costs.

Principle 6. Re-use before Buying, Buy before Building

We will consider re-use of existing applications, systems, and infrastructure before investing in new solutions. We will build only those applications or systems that will provide clear business advantages and demonstrable cost savings

Justification

- Use and availability of effective packaged solutions is increasing.
- Using tested solutions reduces risks.
- Reduces the total cost of ownership.

Implications

- Purchased applications should be compatible with and, where possible, advance the State's technical architecture.
- A Statewide portfolio of strategic applications should be published and maintained.
- Define and adhere to a formal technical review process when evaluating new solutions.

Principle 7. Reengineer First

New information systems will be implemented after business processes have been analyzed, simplified or otherwise redesigned as appropriate.

Justification

- Work processes will be more streamlined efficient and cost effective.
- Work processes, activities, and associated business rules will be well understood and documented.
- Reduces the total cost of ownership.

Principle 8. Total Cost of Ownership

Adopt a total cost of ownership model for applications and technologies which balances the costs of development, support, training, disaster recovery and retirement against the costs of flexibility, scalability, ease of use, and reduction of integration complexity.

Justification

- Leads to higher quality solutions.
- Enables improved planning and budget decision-making.

- Reduces the IT skills required for support of obsolete systems or old standards.
- Simplifies the IT environment.

Principle 9. Minimize Platform Configurations

Create a small number of consistent configurations for deployment across the enterprise.

Justification

- The cost of IT personnel is increasing and the cost of hardware is decreasing rapidly, i.e., "Ride the hardware cost curve".
- This is the most efficient approach to enterprise-wide infrastructure configuration and maintenance.
- By constantly 'tweaking' the performance of an individual server or desktop computer, a multitude of unique configurations is created, thus increasing support and maintenance costs.
- Standardized decisions in product selection simplifies training, learning curve and skills transfer.

Implications

- Platform selection and configuration should support and be coordinated with application requirements and business needs.

Principle 10. Basic Information Services

A standardized set of basic information services (e.g., email, voicemail, e-forms, user training) will be provided to all employees.

Justification

- Increases productivity.
- Reduces costs of maintenance.
- Provides the basis for multi-agency or statewide business initiatives.
- Provides for universal employee access to information.
- Leverages the investments made in technology.

Implications

- Basic information services should be based on, coordinated with, and compatible with application standards.
- Basic information services should be wrapped in standard components and leveraged across the enterprise to facilitate reuse and enhance adaptability.

Principle 11. Anytime/Anywhere Access

Applications, systems, and infrastructure that support the anytime/anywhere access to information and services will be given priority over alternate solutions where practical.

Justification

- Provides maximum value to constituents, partners, and public servants.
- Minimizes deployment costs.
- Makes government services easy to use.

Implications

- Application technologies must provide broad reach through the public Internet, kiosks, and voice recognition.
- The State must develop in-house expertise to support broad-reach technologies.
- The State must make the development of an infrastructure that supports Internet applications and technologies a priority.
- The State should consider using Internet technologies first when developing new applications.
- When buying solutions, priority must be given to applications that leverage the Internet and other customer-friendly interfaces.

Principle 12. Shared Components Using an N-Tier Model

Applications, systems and infrastructure will employ reusable components across the enterprise, using an *n-tier* model.

Justification

- Enables simplification of the environment and geographical independence of servers.
- Takes advantage of modular off-the-shelf components.
- Reuse will lower costs and maintenance efforts.
- Allows for leveraging skills across the enterprise.

Implications

- Requires an organization that administers shared components and promotes reuse.
- Components need to be documented and publicized.
- Requires an enterprise-wide repository standard.
- Applications must be designed to be independent of the User Interface to support access via a variety of sources (e.g. Internet browser, kiosk, voice recognition).

Principle 13. Logical Partitioning and Boundaries

The logical design of application systems and databases should be highly partitioned. These partitions must have *logical boundaries* established, and the logical boundaries *must not be violated.*

Justification

- A change in a database or application can potentially affect many large programs, if they are not highly partitioned.
- Re-coding leads to time-consuming re-testing.

- Partitioning isolates/minimizes change impact.
- Partitioned code is more adaptive to changes in internal logic, platforms, and structures.

Implications

- Publish and use standard application patterns (models) to facilitate design.

Principle 14. Message-Based Interfaces

The interfaces between separate application systems must be *message-based*; this applies to both internal and external systems.

Justification

- The use of messaging is important for enforcing the architecture principle of logical partitioning and boundaries.
- Enables rapid response in maintenance and enhancement activities as required by changes in business processes.
- Messaging technology simplifies integration efforts.
- Messaging technology allows for transparency in locations, databases, and data structures.

Principle 15. Event-Driven Systems

We must deploy application systems that are driven by *business events*.

Justification

- Increases adaptiveness.
- Business processes are a series of business events.
- Business process changes involve the adding, removing, or changing of business events.
- Increases linkage to the business.
- Mirrors the actual business environment.
- Easier to realign IT when change occurs.

Implications

- Requires business analysis expertise (may be best served by establishing a business analyst position).

Principle 16. Physical Partitioning of Processing

We should separate on-line transaction processing (OLTP) from data warehouse and other end-user computing.

Justification

- Separating end-user requests and OLTP maximizes the efficiency of both environments.
- Growth in OLTP is incremental, and requirements are predictable.
- Growth in data warehouses and end-user computing has been nonlinear, and requirements are very difficult to predict.
- Fosters the concept of data stewardship.

Principle 17. Object-Oriented

Application delivery should be evolving toward an object-oriented approach.

Justification

- Objects will allow for easier adaptation of business process changes.
- ISVs (independent software vendors) will progress from components to objects.
- All industry leading application development tools are either object-oriented or object-based.

Implications

- Component development is the logical and practical evolutionary step toward objects.
- A fundamental knowledge of objects must be obtained.
- Object trends must be watched.

Principle 18. Formal Software Engineering

The State shall adopt and employ consistent software engineering practices and methods based on accepted industry standards.

Justification

- Reduces training costs.
- Leads to benchmarks for measurement.
- Enables improved quality assurance.
- Facilitates the reuse of programming modules and code.

Principle 19. Mainstream Technologies

IT solutions will use industry-proven, mainstream technologies.

Justification

- Avoids dependence on weak vendors.
- Reduces risk.
- Ensures robust product support.
- Enables greater use of commercial-off-the-shelf solutions.

Implications

- Niche solutions may require “bleeding-edge” technology where there is no “mainstream” alternative.

Principle 20. Industry Standards

Priority will be given to products adhering to industry standards and open architecture.

Justification

- Avoids dependence on weak vendors.
- Reduces risks.
- Ensures robust product support.
- Enables greater use of Commercial-off-the-Shelf solutions.
- Allows flexibility and adaptability in product replacement.

Implications

- Use of “de facto” industry standards may be necessary.

Principle 21. Disaster Recovery / Business Continuity

An assessment of business recovery requirements is mandatory when acquiring, developing, enhancing or outsourcing systems. Based on that assessment, appropriate disaster recovery and business continuity planning, design and testing will take place.

Justification

- Due to factors such as the Internet and Y2K, customers and partners have heightened awareness of systems availability.
- The pressure to maintain availability will increase in importance. Any significant visible loss of system stability could negatively impact our image.
- Continuation of business activities without IT is becoming harder.
- Application systems and data are valuable State assets that must be protected.

Implications

- May need stress testing tools/labs to ensure adequate system availability.

Principle 22. Enterprise Network as Virtual LAN

We must implement a statewide backbone network that provides a virtual, enterprise-wide local area network.

Justification

- Networks are the essential enabling technology for client/server, Internet, and collaborative computing.
- Knowledge workers' increasing need for access to information across the enterprise. This access must be seamless to reduce decision-making cycle times.
- Lack of a robust network architecture will impact the success of distributed applications.
- Expands the vision of organizations by reaching out to customers and suppliers.

Principle 23. Scalability

The underlying technology infrastructure and applications must be scalable in size, capacity, and functionality to meet changing business and technical requirements.

Justification

- The Total Cost of Ownership is minimized.
- Encourages reuse.
- Leverages the continuing decline in hardware costs.

Implications

- Standards for scalability testing are needed.
- Tactical, time critical, solutions may need to be exempt from this requirement.

Best Practices

The State of Connecticut's Application Architecture is the focal point for its applications systems inventory. It defines how applications should be designed and how they can cooperate to gain the maximum return on investment. It also defines where they execute. The Application Architecture enables:

- Ease of integration of applications and application services.
- Efficient reuse of existing application assets.
- Faster deployment of new applications.
- Better responsiveness to changing business needs.

The best practices listed below provide guidelines for the design or purchase of applications and application components supporting distributed, "thin client" computing for the State of Connecticut.

Best Practice 1: Partition application functionality to mirror business processes

The boundaries between application component functionality should reflect the way work is accomplished in the business unit. Interfaces between components reflect business interfaces so there is linkage between the business and IT solutions.

Rationale:

Adaptive systems are dependent on built-in flexibility and extensibility. By implementing solutions based on the business practices, the impact of future change can be isolated and reduced.

Best Practice 2: Design applications for future usage and added functionality.

Rationale:

Most applications evolve to support new business requirements. Extensibility provides functional scalability.

Best Practice 3: Select best-of-breed Application Development tools in compliance with architecture.

Rationale:

- Historically, project teams selected tools (e.g., Visual Basic, Java) first, and then had to live with the architecture those tools supported. That led to the problem of the tools driving the architecture, and thus the business, rather than the business requirements mandating the tools.
- There is no such thing as one application tool that satisfies all application requirements. Most tools, such as user interface builders, end-user reporting, on-line analytical processing and multi-media, are oriented toward different areas of development.

Best Practice 4: Use an integrated tools set to support the use of the State's formal software engineering practices

Rationale:

- Reduces total cost of ownership.
- Produces higher quality solutions
- Avoids disjointed steps in the software engineering process
- Helps automate proper technical documentation
- Documents metrics that facilitate process improvement

Best Practice 5: The design of all applications shall be documented.

Object models, interaction diagrams and other design artifacts record the structure, behavior and interfaces of application solutions. These are important deliverables of the development process that can benefit future efforts.

Rationale:

- The application design is an asset of the development process, facilitates extensibility and adaptability, and provides for future reuse.

Best Practice 6: Design for the n-tier service oriented architecture.

Rationale

- While many problems inherent in the State's existing monolithic and two-tier applications can be overcome by implementing applications with a three-tier architecture, large, complex projects that are anticipated to have high usage volumes and/or long life spans will be better served by an n-tier service oriented architecture.
- N-tier applications are easily modified to support changes in business rules.
- N-tier applications are highly scaleable.
- An n-tier architecture offers the best performance of any application architecture.
- Any combination of user interfaces (e.g., character, graphical, web browser, and telephone interfaces) may be implemented in an n-tier application.
- N-tier applications are less expensive to build and maintain because much of the code is pre-built and shared by other applications.

Best Practice 7: Design applications that are platform independent.

Rationale

- Designers and operations support staff should make deployment decisions.
- Minimizing platform dependence builds in adaptability and scalability.
- Writing to standard API's protect applications from platform, network and database changes.

Best Practice 8: Generalize application interfaces

- The code providing input and output to the user interface should be designed to provide input and output to as wide a range of interfaces as needed. This should include other applications as well as other types of user interfaces.

- Do not assume that application components will always be accessed via a graphical user interface (or any other user interface).
- Avoid assuming a specific page size, page format, layout language or user language whenever possible.

Rationale

- Generalizing application interfaces facilitates component reuse providing more flexible, scalable solutions.

Best Practice 9: Implement Business Rules As Discrete Components

Rationale:

- Business rules need to be executed to ensure the correct policies are enacted governing the accuracy of related data and the execution of the actions to be performed. By implementing these as discrete components, the users of this information or process can be assured of proper application of the rules.

Best Practice 10: Access data through business rules

Rationale:

- Designing applications so business rules control access to the data assures accuracy, consistency and reliability.
- Data is created and used by business processes. In computer applications, data must be created, used by, and managed by the application component that automates the business process.
- Accessing data in any way other than by business processes bypasses the rules of the module that controls the data. Data is not managed consistently if multiple processes or users access it.
- Federated data should be used wherever possible to assure data accuracy and simplify data management.

Best Practice 11: Assign responsibility for business rules to business units

Assign responsibility for defining and maintaining the integrity of business rules to business units.

Rationale

- IT staff is responsible for coding and administering the software that implements business rules in the network.
- The business units are responsible for the definition and integrity of business rules, and for communicating changes in business rules to IT.
- Every business rule should be assigned to a custodian.

Best Practice 12: Make business rule components platform-neutral

Rationale

- Implement business rules in a non-proprietary, cross-platform language.
- This approach makes platform independence and portability possible.

Best Practice 13: Achieve working system first

Once the detailed application design is complete, concentrate on achieving a working system utilizing off-the-shelf components whenever possible. This will allow the system to be tested first and then optimized later.

Rationale

Early implementation of a working system provides:

- A test platform for proof of concept.
- Validation of the application design early in the development life cycle.
- Early warning of performance issues.
- Early validation or detection of issues provides a greater opportunity to take corrective action.

Best Practice 14: Design for manageability

Design applications so they can be managed using the enterprise's system management practices and tools.

Rationale

Applications and their components require the following management functions:

- Software distribution.
- Start-up, shutdown, and restart of components.
- Starting multiple instances of a component.
- Configuration of components.
- Logging of component operations.
- Communication of errors, exceptions, and unexpected events.
- Security.
- Installation, removal, and update of application modules.
- Version control.

Best Practice 15: Adopt coding standards

Adopt coding standards, in all languages, on all platforms.

Rationale

Coding standards make debugging and maintenance easier. They should address (but not be limited to):

- Naming conventions for variables, constants, data types, procedures and functions.
- Code flow and indentation.
- Error and exception detection and handling.
- Source code organization, including the use of libraries and include files.
- Source code documentation and comments.
- Even the earliest code developed in a project should adhere to the standards.

Best Practice 16: Design for ease of testing

Design application components so they can be tested and debugged easily.

Rationale

- Testing is a critical step in the development of client/server applications.
- Application components with consistent interfaces are easier to test on an application-wide basis.
- Error handling, tracing, and check-pointing should be included.
- These functions should be implemented in the earliest phases of development.

Technical Standards

The State of Connecticut's current application inventory has been documented in the following matrices; each technology item was categorized as follows:

Obsolete - It is highly likely that these standards or products, while still in use, will not be supported by the vendor (industry, manufacturer, etc.) in the future. Some products and standards have already reached the non-supported state. Plans should be developed by the agencies or the State to rapidly phase out and replace them with strategic standards or products. No development should be undertaken using these standards or products by either the agencies or the State.

Transitional - These are standards or products in which an agency or the State has a substantial investment or deployment. These standards and products are currently supported by DOIT, the agencies, or the vendor (industry, manufacturer, etc.). However, agencies should undertake development using these standards or products only if there are no suitable alternatives that are categorized as strategic. Plans should be developed by the agencies or the State to move from transitional to strategic standards or products as soon as practical. In addition, the State should not use these standards or products for development.
Note: many older versions of strategic standards or products fall into this category, even if not specifically listed in a domain architecture document.

Strategic - These are the standards and products selected by the state for development or acquisition, and for replacement of obsolete or transitional standards or products. (Strategic means a three to four year planning horizon.) When more than one similar strategic standard or product is specified for a technology category, there may be a preference for use in statewide or multi-agency development. These preferred standards and products are indicated where appropriate.
Note: some strategic products may be in "pilot testing" evaluation to determine implementation issues and guidelines. Pilot testing must be successfully completed prior to full deployment by the agencies or the State.

Research / Emerging - This category represents proposed strategic standards and products that are in advanced stages of development and that should be evaluated by the State. Some of these standards or products may already be undergoing "hands-on" evaluation. Others will need to be tracked and evaluated over the next 6 to 18 months.

Standards 1: Client Interface Standards (Presentation tier)

The client Interface represents the first tier. Incorporated in the user interface are the menus, the display and the aesthetics of the screens and windows that the user encounters. Presentation is its only task, not processing the business logic.

Due to the need to provide anytime/anywhere access both internally and to the public at large, the State's strategic direction is to deliver applications via a browser-based interface.

Browsers supported for Internet clients are both Microsoft Internet Explorer and Netscape Navigator. Applications should use standard functions and features that can be supported within each browser.

Internal applications may make judicious use of the standard internal browser's features (e.g. DHTML, Java plugins) where it is unlikely that the application will be externalized. Use of a GUI-based interface for some internal applications that require a rich user interface is allowed when a browser-based alternative is not practical.

Supported Client Interface	Obsolete	Transitional	Strategic	Research / Emerging
3270	**X**			
GUI		**X**		
Web User Interface			**X**	
Wireless & Mobile Devices				**X**

In order to expand application availability, the State should continue to explore other application interfaces such as kiosks, interactive voice recognition, and wireless & mobile devices such as the Palm Pilot or Windows CE devices.

Standards 2: Standard Business Tier Languages

Java is the preferred strategic choice based on its multi-platform portability, it's object-orientation, its widespread acceptance, its support for application integration with Microsoft and non-Microsoft legacy applications, its scalability, and its ability to produce reusable components. As the base of shared components increases, new components will be needed less frequently, significantly reducing development costs.

Avoid proprietary Java features or extensions that are provided by the Application Server or IDE Vendors that would prevent an application from running in another J2EE compliant Application Server unless it is the only way to meet the application's business requirements.

All new, custom-developed, Java applications must conform to the State's coding standards and conventions (See Appendix B, Java Coding Standards and Conventions and Addendum A, "Conventions for the Java™ Programming Language" by Sun Microsystems, Inc.)

Following these coding standards and conventions is important because:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

See the middleware domain for supported Java application servers.

Business Tier Languages	Obsolete	Transitional	Strategic	Research/ Emerging
Assembler	**X**			
C++		**X**		
Clipper	**X**			
COBOL	**X**			
COBOL II		**X**		
Delphi		**X**		
Java			**Preferred**	
Oracle Forms (PL/SQL)		**X**		
Power Builder		**X**		
Powerhouse		**X**		
uniVerse Basic		**X**		
Unisys Coolice		**X**		
Unisys Mapper		**X**		
Visual Basic			Acceptable	
Visual C++		**X**		
VB.NET				**X**

Visual Basic is acceptable for workgroup or departmental solutions where enterprise-wide scalability and legacy application integration are not requirements. Business tier solutions using this technology are best deployed using ActiveX DLLs running under Microsoft Transaction Server or COM+.

Standards 3: Standard Data Access Tier

The data access tier is designed to abstract the details of data storage (e.g. schema, stored procedures) from the other application layers. Therefore changes in the database should not require any modification of the code in other tiers in order for them to remain functional.

Supported application languages are the same for the Data Access and Business tiers.

See the Middleware Domain and Data Domain Architectures for details on access methods, supported databases, etc.

Standards 4: Application Architectures

It is important to plan and/or evaluate the architecture of each developed and purchased application to determine the flexibility, extensibility and scalability of the delivered product. Systems that fail to scale appropriately, or are unable to change quickly, are likely to die an early death.

The architecture is a key element to the overall adaptability of an application, and provides many of the services on which the application depends. This includes such capabilities as deploying on an Extranet (i.e. business partner, client), transactional integrity, asynchronous processing, and other system and database level services.

Application Architecture	**Obsolete**	**Transitional**	**Strategic**
Monolithic applications	**X**		
Two-tier applications		**X**	
Three-tier applications		**X**	
N-tier applications using service-oriented application architecture			**X**

Development Guideline:

Parameter and configuration files

Where feasible, use XML documents to persist run-time information.

Benefits

- XML documents are self-documenting for easy readability
- An XML parser will check the file for well-formedness
- If a schema is employed (recommended), an XML parser will perform validation automatically eliminating custom validation code
- The parameter file can be edited with validation by a shrink-wrap XML editor
- Additional items may be added to the file without the need to modify all readers of the file

Standards 5: Inter/Intra Application Communication

Messaging is the preferred method for passing information between application tiers. Use of State-standard message oriented middleware for inter-application communication will provide a mechanism that is open to any platform, any language, and any object framework.

The standard Inter-application message format is the XML document as defined by XML schema. (Consider using XML documents for intra application communication as well when adaptability is more important than performance or disparate technologies are used to implement application tiers.)

Benefits:

XML documents are self-defining, which allows new information to be added without having to modify all applications that use the document, facilitating change.

XML documents can easily be reused or extended.

Use of XML documents facilitates development by reducing custom validation code and by pushing much of the integration testing back into unit testing (validation against an XML Schema reduces errors in message formats that would otherwise only be caught in more costly integration testing).

XML documents are becoming the defacto standard in the industry for business to business communication, enterprise application integration, reporting, presentation, and content management.

Use of XML documents as the message format for passing information from the middle application tiers to the presentation tier allows the information to be transformed (using XSLT and/or CSS) to a variety of different formats (e.g. HTML, WML). These transformations allow the same information of be viewed in a variety of ways (e.g. browsers, mobile devices, text only) by only modifying the presentation tier.

Web Services (UDDI, SOAP) are emerging industry standards for sending XML documents that should be considered for inter-application communications. *See the middleware domain for more information.*

Inter/Intra Application Communication	**Obsolete**	**Transitional**	**Strategic**	**Research / Emerging**
Web Services				**X**
SOAP				**X**
XML 1.0			**X**	
W3C XML Schema			**X**	
XSLT 1.0			**X**	

Standards 6: Web Development Standards

Active Server Pages (ASP) with ActiveX components:

Within the State this technology has been primarily implemented using Visual InterDev, Visual Basic, and Visual C++. Visual InterDev provides an Integrated Development Environment (IDE) that assists developers in adding VBScript to HTML pages, allowing the creation of dynamic server-side content as well as providing the "glue" between the user interface and the business components. The business components (ActiveX DLLs) are primarily produced using Visual Basic extended with Visual C++ to add functionality that is not available using Visual Basic alone. Active Server Page applications run mainly under Microsoft Internet Information Server with the ActiveX components running under Microsoft Transaction Server or COM+.

Java Server Pages (JSP) with Servlets and Enterprise Java Beans:

The trend in developing n-tier Java applications is toward server-side development of Enterprise Java Beans (EJB) and thin-client GUIs using Java Server Pages. This technology is analogous to Active Server Pages. Developers separate presentation logic from business components by encapsulating business/application logic in Beans that are accessed through tags that content creators can use.

Supported Web Technology	**Obsolete**	**Transitional**	**Strategic**	**Research / Emerging**
Active Server Pages / ActiveX Components		**X**	Acceptable	
ASP.NET				**X**
CGI		**X**		
Cold Fusion		**X**		
HTML			**X**	
Java Server Pages / Servlets			**Preferred**	

The preferred strategic direction for web application development is use of Java Server Pages and Servlets. This technology has the advantage of multi-platform availability, good integration with Microsoft and non-Microsoft legacy systems, robust scalability, and full object orientation.

Java Server Pages vs. Active Server Pages Comparison	
Java Server Pages	Active Server Pages
Runs on all major web servers	**Runs primarily on Microsoft IIS**
Is interpreted only once to Java byte-code, and re-interpreted only when the file is modified	**Is interpreted each instance**
Provides better facilities for separation of page code and template data by means of JavaBean, Enterprise JavaBeans and custom tag libraries	**Uses COM/DCOM to call compiled components within DLLs. DLL changes require a server reboot which inhibits this practice in a production environment.**

Use of Active Server Pages and ActiveX components is an acceptable strategic alternative subject to the following guidelines:

Web Project Type	ASP/ActiveX	JSP/Servlets
Work Group	Acceptable	**Preferred**
Departmental	Acceptable	**Preferred**
Legacy Integration	**Not Acceptable**	Acceptable
Enterprise-wide w/limited scalability and/or availability	Acceptable	**Preferred**
Enterprise-wide w/ high scalability and/or availability	**Not Acceptable**	Acceptable
Public w/limited scalability and/or availability	Acceptable	**Preferred**
Public w/ high scalability and/or availability	**Not Acceptable**	Acceptable

Java / Visual Basic Integration:

The downside of supporting both Microsoft and Java technologies is the need to integrate the technologies. This can be accomplished using messaging as in Inter/Intra Application Communication above. One of several Java/COM bridges also can be used. An emerging alternative to the Java/COM bridge is the Web Service, which may become the future integration method of choice.

Deployment Guidelines:

Internet Applications

All browser-based applications should support the major public browsers (e.g. Internet Explorer, Netscape). Applications should be developed using server-side processing (e.g.

Servlets or ActiveX DLLs) for all but rudimentary edits (using embedded JavaScript run within the browser). Browser-specific extensions (e.g. VBScript in Internet Explorer) and required plug-ins (e.g. JRE) should be avoided.

Internet applications are typically deployed in special "demilitarized zones" (DMZs) delineated by Firewalls to protect the State from unauthorized access (hacking) and malicious computer programs. It is vital that applications, which will be deployed in this hazardous environment, be designed to adhere to the required security restrictions. It is good practice to become familiar with these security restrictions prior to designing the architecture of an application.

Intranet/Extranet Applications

Internal applications may make assumptions based on internal browser standards in order to extend or enrich an application (e.g. use of client-side VBScript or Applets), but only when such additional functionality is required to support business requirements that could not be reasonably supported using only server-side processing. The possibility that an internal application may need to be extended to partners (via Extranet) that may not adhere to the State's architectural standards should be considered.

Java Web Start is an acceptable alternative to the browser for providing a Java Runtime Environment for Java applications for frequent users of an application. This technology provides a solid, zero administration environment for Java applications or applets. Once WebStart is installed, it provides a self-updating mechanism (from an HTTP server) for itself, the Java Runtime Environment(s) and the application(s) it supports. The advantage over an applet running in a browser is that applications are cached locally improving startup time. Applications are downloaded when an application is updated and not every time it is invoked. Occasional users of an application are best served by running the application within the browser.

It is recommended that Oracle Forms development be phased out over time in favor of another EWTA approved development environment. For agencies with an existing investment in Oracle Forms 6i, it is a viable front end for applications designed to use an Oracle database as the backend. This RAD tool allows the developer direct access to exploit the power of the Oracle database. PL/SQL is used to code within the form and to create shared modules that can be plugged into many different 6i forms. Oracle Forms 6i applications, used in conjunction with Oracle 9iAS, can be deployed via a web browser as a Java applet without requiring any special programming or additional processing. Deployment requires the Jinitiator browser plug-in or Internet Explorer 6 or above.

	Internet Applications		Intranet Applications	
Supported Internet Technology	Client Side	Server Side	Client Side	Server Side
ActiveX Components	**Not recommended**	Acceptable	**Not recommended**	Acceptable
VBScript	**Not recommended**	Acceptable	Acceptable (Internet Explorer only)	Acceptable
JavaScript	Acceptable	**Not recommended**	Acceptable	**Not recommended**
HTML	Recommended	Recommended	Recommended	Recommended
Jscript	**Not recommended**	**Not recommended**	**Not recommended**	**Not recommended**
XML	Recommended	Recommended	Recommended	Recommended
Applets	**Not recommended**	**N/A**	Acceptable (Requires JVM)	**N/A**
Servlets	**N/A**	Recommended	**N/A**	Recommended
Web Start hosted Java Applications	**Not recommended**	**N/A**	Acceptable	**N/A**

Integrated Development Environments (IDEs)

Use of IDE supports the design/code/deploy/debug cycle of modern development. An integrated environment is designed to enhance developer productivity by making the mechanics of development easier and faster.

Selection of an IDE is tightly bound to the development environment for which it is designed.

In the case of Microsoft technology, use of Visual Studio Enterprise Edition is an easy pick to support Active Server Page development due to limited competition and an existing base in the State.

Choosing an IDE to support Java Server Page development is more difficult due to the presence of several strong players (i.e. BEA, IBM, Inprise, Oracle). Meta Group recommends that Java IDE selection be coupled with application server selection due to the industry trend to integrate one with the other for maximum productivity and functionality.

Web IDE	Obsolete	Transitional	Strategic	Research / Emerging
Cold Fusion		X		
Oracle Forms		X		
Oracle JDeveloper				X
JBuilder				X
Visual Age Java Enterprise Edition / Websphere Studio			X	
Visual Café			X	
Visual Studio (Visual Interdev / Visual Basic / Visual C++*)			X	
Visual Studio.NET				X
XML SPY 4.0				X

* Note: Visual C++ is only to be used only to augment Visual Basic development when Visual Basic does not support a required feature.

Standards 7: Application Security Standards

See the Application Security Guidelines in Appendix A.

Standards 8: Standard Project Management Tools

Use of project management principles is crucial to successful project delivery. Due to its popularity, MS Project has become the *de facto* State standard. However, a more robust tool set with enterprise-wide functionality should be considered.

Supported Tools	Obsolete	Transitional	Strategic	Research / Emerging
MS Project			X	
PlanView PSA				X

Standards 9: Configuration (Source Code) Management

Code management is crucial to maintain application integrity through the development and

Supported Tools	Obsolete	Transitional	Strategic	Research / Emerging
Pan Valet		**X**		
Rational ClearCase				**X**
SCLM		**X**		
Visual Source Safe		**X**		

maintenance cycle. Ideally code management tools should integrate with defect tracking and project build tools. The State will require a code management system that can scale across the enterprise to foster an environment that supports re-use of shared components across the enterprise. We do not have enough information at this time to recommend a strategic choice in modeling tools, but Rational ClearCase is being considered.

Standards 10: Object Modeling Tools

Use of object modeling principles and supporting tools are crucial to the successful implementation of large-scale object-oriented applications. We recommend modeling tools that support industry standard UML. Ideally the tool integrates with the developer's IDE enabling roundtrip engineering between class diagrams and the code and provides roundtrip engineering.

Object Modeling Tools	Obsolete	Transitional	Strategic	Research / Emerging
Rational Rose				**X**
Together J				**X**
Visual Modeler		**X**		

Standards 11: Enterprise Reporting / Structured Information Delivery

Internet-capable report writers are a vital part of the State's strategy to provide anytime/anywhere access to information. These tools are designed to enable developers to easily deliver reports that are either fundamental to a system or more complex than the end-user can reasonably be expected to produce.

Products in this class provide browser-based report viewing, interactive drill down and dynamic sorting of report data via DHTML. Data download options are built in (e.g. spreadsheets, XML documents, Acrobat files).

Enterprise Reporting tools are primarily designed for electronic information dissemination, but are also suitable for medium to large paper reporting requirements where batch scheduling and unattended operation are desired.

Enterprise Reporting	Obsolete	Transitional	Strategic	Research / Emerging
Actuate e.Reporting Suite 5				X
Crystal Enterprise 8.x			X	
Crystal Professional 8.x		X		
InetSoft Style Reports 4.x			X	
Easytrieve Plus		X		
Focus		X		
MS Access*		X		
QMF		X		

*Although we do not recommend Microsoft Access as a strategic developer-reporting tool for enterprise-wide technical architecture, it does provide a single end-user solution for desktop or small workgroup applications. MS Access is also a useful tool for rapid development of application prototypes and database design prototypes. A MS Access prototype database can be readily upsized to more strategic databases via add-ins (e.g. Scriptoria for UDB) that create DDL and export the data.

Java-based reporting tools:

InetSoft Style Reports was chosen to support the Java development environment based on its robust feature set, its installed base within the State, and its favorable value to cost ratio. InetSoft is a small privately held company and its long-term viability needs to be monitored to mitigate this risk.

Actuate e.Reporting Suite is a strong contender supported by a viable vendor, but its high entry price and lack of an installed base within the State keep it from being classified as strategic at this time. This product's robust feature set and support of both the Java and Microsoft development environments warrant monitoring for future consideration.

Microsoft-oriented reporting tools:

Crystal Decisions offers two products capable of Enterprise Reporting (ER),

Crystal Reports Professional and Crystal Reports Enterprise.

- Close analysis of Crystal Professional revealed that its fundamental design is not suited for ER without adding additional, costly, components. While not meeting the state's standards for ER, Crystal Professional is acceptable for shops with an existing investment in this technology for developer-oriented report writers in cases where the "Internet-capable" requirements are minimal to none as it can readily be migrated to Crystal Enterprise should the requirements change.

- After thorough evaluation including TCO, Crystal Enterprise emerged as the preferred solution for non-Java development.

Standards 12: Ad Hoc Query/Analysis

In situations where the drill down and sorting features of Enterprise Reporting products are insufficient to meet user needs for custom reporting, Ad Hoc Query / Analysis tools may be used to satisfy the needs of the "advanced" user.

These tools provide a less rigorous development environment that do not necessarily require the services of a professional developer to use effectively while providing ad hoc query, user-developed reports, and access to OLAP cubes.

Supported Tools	Obsolete	Transitional	Strategic	Research / Emerging
Brio Intelligence				X
Cognos PowerPlay				X
Business Objects				X
Crystal Analysis Professional				X
MS Access		X		
SAS*		X		
SPSS*		X		

The top industry contenders are listed as Research items for future consideration. The line between Enterprise Reporting and Ad Hoc Query and Analysis tools continues to blur as vendors extend their products features. Similarly, some tools in this category (e.g. Cognos PowerPlay) have multidimensional features that may satisfy modest OLAP requirements.

- Statistical analysis products – no strategic recommendation available.

Standards 13: Online Analytical Processing (OLAP) tools

Supported Tools	Obsolete	Transitional	Strategic	Research / Emerging
Hyperion Essbase				X
Cognos PowerPlay				X
Oracle Express				X
MicroStrategy				X

Tools in this category are required when high-end scalability and advanced ad hoc analytical queries and cross-dimensional operations are required. First tier products is this category are listed for future reference.

Standards 14: Geographic Information Systems (GIS) Software Standards

Development of GIS based applications plays a critical role in providing and maintaining spatial information to the public and decision-makers. Several State agencies have invested

Supported Tools	Obsolete	Transitional	Strategic	Research / Emerging
ESRI ArcGIS Desktop 8.x (Includes ArcView 8.x and ArcInfo 8.x)			Preferred	
Intergraph GeoMedia Professional 5.x			Acceptable	
MapInfo Professional 7.x		X		
ESRI ArcView 3.x		X		
ESRI ArcInfo 7.x		X		
ESRI ArcPad 6.x			X	
ESRI ArcIMS 4.x			X	
ESRI ArcIMS 3.x		X		

considerable amount of resources in delivering products based on GIS technologies.

Product Selection and Upgrades:

First time users are advised to use the preferred ESRI products, however, we understand that rationale may exist based on the business needs to use the Acceptable Intergraph GeoMedia Professional product. Users are welcome to request consultation with the GIS subcommittee members for making the proper choice.

Agencies that are already using ESRI products such as older versions of ArcView GIS or ArcInfo (i.e. versions prior to 7.x) should upgrade to ESRI ArcGIS Desktop 8.x. Similarly, if the agencies are using older version of Intergraph Geomedia Professional, then they should upgrade to the most recent version of the Intergraph GeoMedia Professional or switch over to ESRI products. Agencies currently using Mapinfo software should transition to ESRI ArcGIS 8.x product.

ArcView GIS 3.x and Arc/Info 7.x to ArcGIS 8.x conversion issues:

Refer to "ArcGIS Migration Guide"[1] in Addendum B for detailed discussion on the topic.

---

[1] ArcGIS Migration: Written for GIS Managers is reprinted courtesy of ESRI. Copyright (c) 2001 ESRI. All rights reserved.

Integration and the future of GIS:

Companies making GIS based software have relied on proprietary design and code, however, currently GIS software vendors have formed the OpenGIS Consortium and are working on establishing and developing OpenGIS specifications. The OpenGIS specifications will enable users and developers to freely exchange and apply spatial information, applications, and services across networks, different platforms and products.

Application Development:

When a vendor's product offers the option, applications should be developed using EWTA-compliant languages (e.g. Java, Visual Basic).

To be determined:

Standards

- Object Modeling Standards and Tools (agency suggestion: Mega Suite)
- Standard Development Methodology Documentation Tools and Standards
- Source Code Management Standard (agency suggestion: Star Team)
- Consideration of GUI front-end tools for legacy applications (agency suggestion: Seagull, Relativity)

Guidelines

- Component Development Guidelines
- Design Pattern Library Guidelines
- Software Metrics Standards
- J2EE / DCOM Integration Guidelines
- Software package Integration Guidelines
- Business Process Reengineering Guidelines
- Inter/Intra Application Communications (XML) Guidelines
- Quality Assurance and Quality Control Guidelines
- Java Framework

Other

- Standard Defect Tracking System
- Statewide Component Library